Design Decisions:

Subclass Structure

* Decided on implementing an abstract Entity class that contains all Plants and Zombies as they share many attributes(Health, Attack, Attack Speed, etc)
* While only one zombie exists now, since more will be added in the future, we created an Abstract zombie class from which our BasicZombie is a subclass.

Level Grid Structure

* To maintain scalability and organization, each level is comprised of lanes. These lanes act almost as independent games from each other since plants and zombies in one lane have no impact on the plants and zombies in another.
* Lanes are further split into spots. Each spot is able to contain up to a single plant (Those close to the right side are typically prevented from hosting plants)
* While Spots are responsible for storing the plants on the game board, since zombies can move, storing them in spots wouldn’t make sense. As a result, zombies are stored within an ArrayList in the lane objects. Initially, a queue would make more sense to store zombies, given that we could easily find out which zombies are in front. In future iterations, zombies will have different movement speeds and be able to pass one another, making this structure meaningless.
* The turn functionality was broken down into each of our classes (The controller calls turn on Level, level calls turn on each lane and lane calls turn on each plant and zombie).

MVC Model

* Changed all code to follow the MVC model to allow for easy development process.
* Original classes from previous milestone leading up to main have been added to model package. They make up the contents of the application.
* Main class from previous milestone split into three classes, Controller, GameCanvas, and Main. Controller handles user input, calls all methods needed to play the game, and updates the View. GameCanvas is the basic look of the level. Main starts the game.
* View class has been added to the view package. It displays the graphical representation of the model, showing where the plants have been placed and where the zombies are.
* Controller communicates the user input to the model, where all the backend damage, movement, sun dollars, and so on are calculated. This information is sent to the view, and the view updates accordingly. Then the code returns back to the controller so the user can make their next input.

Design Changes(M2->M3)

* Originally, zombies had a movespeed recorded in terms of the number of pixels they would cross per turn, this proved difficult to deal with when working with computers of different resolutions. As a result, this was changed so that instead zombie’s movespeeds are in terms of the number of spots they can traverse per turn(ie:0.8, 1.0, etc). This allows us to deal with all the screen resolution issues over in our gamecanvas rather than in our zombie class. This also allowed us to change how our lanes recorded distance, using only the number of spots rather than the number of pixels we were working with.
* Rather than hard coding the size of each of our views components, they were changed to be dependent on the size of the screen being played on, that way the game should work regardless of screen resolution.
* How the Plants are dealt with in the controllers and how plant buttons were created has also been changed during this milestone, using a dynamic approach that doesn’t need to be changed upon the addition of new plants.
* The undo redo feature was added using a doubly linked list, storing past plant placements, as well as future plant placements(Which only exist after undoing). Whenever a plant was placed, the future plants were cleared, and the plant was added to the past placements. At the end of each turn, the entire linked list was cleared.

Design Changes(M3->M4)

* Added save/load using GSON. The GSON jar file is imported using Maven. Our saveGame method converts the level (containing all entities) object to JSON data. The loadGame method reads the JSON data and converts it back to the level object.
* Added levelbuilder. Levels are created using a GUI with a list of buttons (one per lane) that allows players to place a zombie of their choosing in that lane. Once all zombies have been added for that “wave”, the user can press next to choose the next wave of zombies. When the user is finished, they can press done and the level is created.
* Added mainmenu gui which is launched from main method. In the main menu the user can select whether they would like to start a new game, or load from a dropdown menu of saved levels.

Project Process Reflection

* Overall we liked our design process. We created a rough UML diagram of the overall structure of the code at the beginning of the project, and have stuck to it. All plants and zombies are entities. Entity and space inherit from level, and level is used by model. From there the MVC model was implemented as discussed in class.
* Using GSON made serializing much easier. At one point we were considering iterating through every space and saving the entity on it, then parsing for every possible entity on that space. This idea would have been incredibly inefficient and time consuming to code.
* We are also very happy with our implementation of the MVC model. Using MVC allowed us to have a much faster development process. We were able to divide up the work, without the worry of merge conflicts because each group member could work on a separate package. It also ensured our work was extremely easy to organize. Using MVC also prevented much code duplication by allowing us to create multiple views.
* Creating UML diagrams was also very beneficial when trying to get group members up to speed on our changes. When explaining the changes, a group member could simply point at the UML diagram as opposed to searching through every package in the code.
* The implementation of undo/redo was also superb. A linkedlist was used to easily store the order of the plants placing. From there, previously existing methods such as killPlant and addPlant were used to “undo” and “redo” the moves. Almost no extra code was needed to create these methods
* One thing that could have been improved is our adherence to the real Plants vs Zombies game rules. For example, in real game, after placing a plant there is a cooldown on when you can place the next plant. We could have implemented a turn delay to mimic this.
* Another thing that could have been added is a real menu to give instructions, as opposed to a pop-up.
* We should have tried to port each version of our code to Android, as opposed to only porting the finished product. By leaving it to the last milestone, we did not have enough time to figure out how the Android dev platform worked. If we had learned it over the period of milestones 1-4 we definitely could have tried to get the bonus 5%

What Smells:

* While for the most part, our code is relatively smell free, the Lane.java class has started to accumulate some functions that serve very specific purposes and have made it a little more “spaghetti-like”. This will be addressed as we work on milestone 4.

Design Decisions Authors: Simon Krol and Shaun Gordon